Efficient Neural Neighborhood Search for Pickup and Delivery Problems

Yining Ma[1,*] , **Jingwen Li**[1,*] , **Zhiguang Cao**[2,†] , **Wen Song**[3] , **Hongliang Guo**[4] , **Yuejiao Gong**[5] and **Yeow Meng Chee**[1]
[1]National University of Singapore
[2]Singapore Institute of Manufacturing Technology, A*STAR
[3]Institute of Marine Science and Technology, Shandong University
[4]Institute for Infocomm Research, A*STAR
[5]South China University of Technology
{yiningma, lijingwen}@u.nus.edu, zhiguangcao@outlook.com, wensong@email.sdu.edu.cn, guo_hongliang@i2r.a-star.edu.sg, gongyuejiao@gmail.com, ymchee@nus.edu.sg

Abstract

We present an efficient Neural Neighborhood Search (N2S) approach for pickup and delivery problems (PDPs). In specific, we design a powerful Synthesis Attention that allows the vanilla self-attention to synthesize various types of features regarding a route solution. We also exploit two customized decoders that automatically learn to perform removal and reinsertion of a pickup-delivery node pair to tackle the precedence constraint. Additionally, a diversity enhancement scheme is leveraged to further ameliorate the performance. Our N2S is generic, and extensive experiments on two canonical PDP variants show that it can produce state-of-the-art results among existing neural methods. Moreover, it even outstrips the well-known LKH3 solver on the more constrained PDP variant. Our implementation for N2S is available online[1].

1 Introduction

Efficient neighborhood search functions as the key component of powerful heuristics for pickup and delivery problems (PDPs) [Parragh *et al.*, 2008]. It involves an iterative search process which transforms a solution into another candidate in its current neighborhood, hopefully in an efficient way. Designing the neighborhood and search rules usually determines the efficiency and even the success of a solver. However, they are often problem-specific and *manually engineered* with a lot of trial and error, which needs to redesign when changes occur in constraints or objectives. These limitations may hinder the applications in the rapidly evolving industries.

On the other hand, recent neural methods for vehicle routing problems (VRPs) have emerged as promising alternatives to traditional heuristics (e.g., [Ma *et al.*, 2021b]). They are usually faster, and more importantly, could automate the design of heuristics for new variants where no hand-crafted rule is available [Kwon *et al.*, 2020]. However, the prevailing neural methods mainly focus on travelling salesman problem (TSP) or capacitated vehicle routing problem (CVRP), where efficient solvers for PDPs are rarely studied. The PDP is ubiquitous in logistics, robotics, meal-delivery services, etc [Parragh *et al.*, 2008], which optimizes the route for pickup-delivery requests and is characterised by the precedence constraint (pickup before delivery). Despite the first attempt in [Li *et al.*, 2021b], which learns a construction method to build a PDP solution (route) in seconds, it leaves a considerable gap to traditional heuristics in solution quality.

To reduce the gap, we propose an efficient *Neural Neighborhood Search (N2S)* approach for PDPs, based on a novel Transformer styled policy network with the encoder-decoder structure. To our knowledge, the most similar existing policy network to ours is the DACT in [Ma *et al.*, 2021b]. Also as an improvement method, DACT learns to encode the current solution and transform it into another one using the *2-opt*, *insert*, or *swap* decoder. Among them, the *2-opt* decoder, which considers reversing a segment of the solution, performed the best for TSP and CVRP. However, it is not suitable for PDPs since the precedence constraint can be easily violated by segment inversion. Though the *insert* decoder performed well on small-scale PDPs, which considers removing and reinserting a single node, its performance significantly drops on larger-scale or highly constrained PDPs (see Section 5). Conversely, our N2S tackles the precedence constraint more efficiently by allowing a pair of pickup-delivery nodes to be simultaneously operated in the neighborhood search through two customized *removal* and *reinsertion* decoders as shown in Figure 1.

Another challenge lies in the design of the encoders. It was well revealed in [Ma *et al.*, 2021b] that the vanilla Transformer encoder [Vaswani *et al.*, 2017] failed to correctly encode route solutions since the embeddings of node features (i.e., coordinates) and node positional features (i.e., node positions) involve two different *aspects* of a route solution which are not directly compatible during encoding. They thus proposed the dual-aspect collaborative attention (DAC-Att) to learn dual representations for each feature *aspect*. In

*Equally contributed.

†Zhiguang Cao is the corresponding author.

[1]Code is available at https://github.com/yining043/PDP-N2S. Please refer to https://arxiv.org/abs/2204.11399 for an extended version of this paper with full results of Table 7 and more discussion.

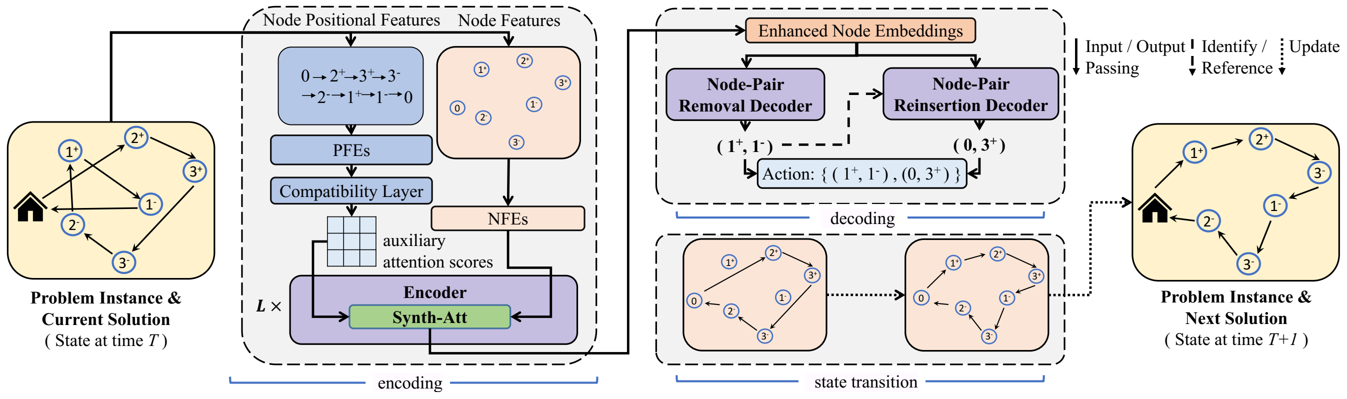

Node Positional Features
Node Features
PFEs
Compatibility Layer
NFEs
auxiliary attention scores
Encoder
Synth-Att
L ×
encoding
Enhanced Node Embeddings
Node-Pair Removal Decoder
Node-Pair Reinsertion Decoder
Action: { (1+, 1-) , (0, 3+) }
decoding
state transition
Input / Output Passing
Identify / Reference
Update
Problem Instance & Current Solution (State at time T)
Problem Instance & Next Solution (State at time T+1)

Figure 1: An example of a PDTSP-7 instance to illustrate our N2S approach. From the left to the right: encoding, decoding and state transition process. The encoders process raw features of the current solution to produce node embeddings, which are then fed into the two decoders to sample an action. In the state transition, the node pair $(1^+, 1^-)$ are removed and then reinserted after depot (0) and node 3^+, respectively.

this paper, we propose a simple yet powerful *Synthesis Attention (Synth-Att)* where the attention scores from various types of node feature embeddings can be synthesized to attain a comprehensive representation. It not only has the potential to encode more *aspects* than DAC-Att, but also reduces the computation costs while reserving competitive performance.

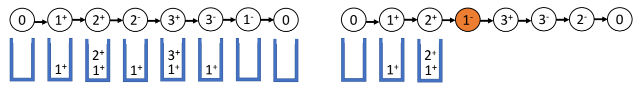
Additionally, we design a diversity enhancement scheme to further ameliorate the performance. The proposed N2S approach was trained through reinforcement learning [Ma *et al.*, 2021b], and we evaluated it on two canonical problems in the PDP family, i.e., the pickup and delivery travelling salesman problem (PDTSP) and its variant with the last-in-first-out constraint (PDTSP-LIFO) to verify our design. Experimental results show that our N2S outperforms the state-of-the-art, and becomes the *first* neural method to surpass the well-known LKH3 solver [Helsgaun, 2017] when solving the synthesized PDP instances.

The contributions of the paper are summarized as follows: 1) we present an efficient N2S approach for PDPs, which learns to perform removal and reinsertion of a pickup-delivery node pair automatically; 2) we propose Synth-Att, which allows the vanilla self-attention mechanism to synthesize node relationships from various feature embeddings in a simple way, and achieves superior expressiveness with much fewer computation costs in comparison to DAC-Att; 3) we explore a diversity enhancement scheme, which further makes our N2S the *first* neural method with almost no domain knowledge to surpass the LKH3 solver on the synthesized PDP instances (when there are no significant distribution shift w.r.t training ones). All these advantages highlight the potential of N2S as a powerful neural solver for new PDP variants without much need for manual trial and error.

2 Related Work

Neural Methods for VRPs. We classify recent *neural* methods into *construction* and *improvement* ones. The *construction* methods, e.g., [Nazari *et al.*, 2018; Joshi *et al.*, 2019; Kool *et al.*, 2018], learn a distribution of selecting nodes to autoregressively build solutions from scratch. Despite being fast, they lack abilities to search (near-)optimal solutions, even if armed with sampling (e.g., [Kool *et al.*, 2018]), local search (e.g., [Kim *et al.*, 2021]), or Monte-Carlo tree search (e.g., [Fu *et al.*, 2021]). Among them, POMO [Kwon *et al.*, 2020] which explored diverse rollouts and data augments is recognized as the best construction method. Differently, *improvement* methods often hinge on a neighborhood search procedure such as *node swap* in [Chen and Tian, 2019], *ruin-and-repair* in [Hottung and Tierney, 2020], and *2-opt* in [Wu *et al.*, 2021]. [Ma *et al.*, 2021b] extended the Transformer styled model of [Wu *et al.*, 2021] to Dual-Aspect Collaborative Transformer (DACT), and achieved the state-of-the-art performance, which was also competitive to the *hybrid* neural methods, e.g., the ones combined with differential evolution [Hottung *et al.*, 2021] and dynamic programming [Kool *et al.*, 2021]. Despite the success of the above methods for CVRP or TSP, they are not verified on the precedence constrained PDPs. Though [Li *et al.*, 2021b] made the first attempt to learn a construction solver for PDPs by introducing the heterogeneous attention to [Kool *et al.*, 2018], the resulting solution qualities are still far from the optimility.

Neighbourhood Search for PDPs. Various heuristics based on neighborhood search have been proposed for PDPs. For PDTSP, [Savelsbergh, 1990] studied the k*-interchange* neighborhood. A *ruin-and-repair* neighborhood was later proposed in [Renaud *et al.*, 2000], and further extended in [Renaud *et al.*, 2002] with multiple perturbation methods. Aside from PDTSP, other PDP variants were also studied, normally solved by designing new problem-specific neighborhoods [Parragh *et al.*, 2008], e.g., [Veenstra *et al.*, 2017] proposed five neighborhoods to tackle the PDTSP with handling costs. Among them, the PDTSP-LIFO attracts much attention due to its largely constrained search space. To solve it, [Carrabs *et al.*, 2007] introduced additional neighborhoods such as *double-bridge* and *shake*. In [Li *et al.*, 2011], neighborhoods with tree structures were further proposed. Different from the above PDP solvers, the well-known LKH3 solver [Helsgaun, 2017] combines neighborhood restriction strategies to achieve more effective local search, which could solve various VRPs with superior performance. Recently, LKH3 was extended to tackle several PDP variants including PDTSP



(a) δ_1 (feasible to both PDPs) (b) δ_2 (only feasible to PDTSP)

Figure 2: Two PDP solutions. (a) all goods are on top of the stack at the delivery node; (b) goods from 1^+ is blocked by 2^+ at 1^-.

and PDTSP-LIFO, and delivered comparable performance to [Renaud *et al.*, 2002] and [Li *et al.*, 2011] on PDTSP and PDTSP-LIFO, respectively. Also given the open-sourced nature[2], we use LKH3 as the benchmark heuristic in this paper.

3 Problem Formulation

3.1 Notations

We define the studied PDPs over a graph $G = (V, E)$, where nodes in $V = P \cup D \cup 0$ represent locations and edges in E represent trips between locations. With n one-to-one pickup-delivery requests, an PDP instance contains $|V| = 2n + 1$ different locations, where node 0 is depot, set $P = \{1^+, 2^+, ..., n^+\}$ contains pickup nodes, and set $D = \{1^-, 2^-, ..., n^-\}$ contains delivery nodes[3]. Each pickup node i^+ has a number of goods to be transported to its delivery node i^-. The objective is to find the shortest Hamiltonian cycle to fulfill all requests. In this paper, we consider two representative PDP variants, i.e., PDTSP and PDTSP-LIFO. The solution δ is defined as a cyclic sequence $(x_0, ..., x_{2n+1})$, where x_0 and x_{2n+1} are the depot, and the rest is a permutation of nodes in $P \cup D$. For PDTSP, such permutation is under the *precedence* constraint that requires each pickup i^+ to be visited before its delivery i^-. For PDTSP-LIFO, the *last-in-first-out* constraint is further imposed which requires loading and unloading to be executed in the corresponding order. This implies that unloading at a delivery node is allowed if and only if the goods is at the top of the stack. In Figure 2, we present two example solutions with $n = 3$ and $|V| = 7$. The two solutions are both feasible to PDTSP, however, solution δ_2 in Figure 2(b) is infeasible to PDTSP-LIFO as the goods from 1^+ is *NOT* at the top of the stack when it needs to be delivered at 1^-.

3.2 MDP Formulations

We define the process of solving PDPs by our N2S as a Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ as follows.

State $\mathcal{S}$. At time t, the state is defined to include: 1) features of the current solution δ_t, 2) action history, and 3) objective value of the best incumbent solution, i.e.,

$$s_t = \{\{l(x)\}_{x \in V}, \{p_t(x)\}_{x \in V}, \mathcal{H}(t, K), f(\delta_t^*)\}, \quad (1)$$

where δ_t is described from two *aspects* following [Ma *et al.*, 2021b]: $l(x)$ contains 2-dim coordinates of node x (i.e., *node features*) and $p_t(x)$ indicates the index position of x in δ_t (i.e., *node positional feature*); $\mathcal{H}(t, K)$ stores the most recent K actions at time t if any; and $f(\cdot)$ denotes the objective function and $\delta_t^* = \arg\min_{\delta_{t'} \in \{\delta_0, ..., \delta_t\}} f(\delta_{t'})$.

[2]http://webhotel4.ruc.dk/~keld/research/LKH-3/

[3]We also refer x_0 to the depot, $\{x_1, ..., x_n\}$ to the pickup nodes, and $\{x_{n+1}, ..., x_{2n}\}$ to the delivery nodes from here on.

Action $\mathcal{A}$. With action $a_t = \{(i^+, i^-), (j, k)\}$ where $j, k \in V \backslash \{i^+, i^-\}$, the agent removes node pair (i^+, i^-), and then reinserts node i^+ and i^- after node j and k, respectively.

State Transition $\mathcal{T}$. We utilize a deterministic transition rule to perform a_t. An example is illustrated in Figure 1.

Reward $\mathcal{R}$. The reward function is defined as $r_t = f(\delta_t^*) - min\left[f(\delta_{t+1}), f(\delta_t^*)\right]$ which is the immediate reduced cost w.r.t. $f(\delta_t^*)$. The N2S agent aims to maximize the expected total reduced cost w.r.t. δ_0 with a discount factor $\gamma < 1$.

4 Methodology

4.1 Encoder and Synth-Att

Given state $s = \{\{l(x)\}_{x \in V}, \{p(x)\}_{x \in V}, \mathcal{H}(t, K), f(\delta^*)\}$, the N2S encoder takes $\{l(x)\}_{x \in V}$ and $\{p(x)\}_{x \in V}$ as inputs[4] to learn embeddings for representing the current solution. Following DACT, we first project these raw features into two sets of embeddings, i.e., node feature embeddings (NFEs) $\{h_i\}_{i=0}^{|V|}$ and positional feature embeddings (PFEs) $\{g_i\}_{i=0}^{|V|}$. Different from [Ma *et al.*, 2021b], we treat NFEs as the primary set of embeddings whereas PFEs as auxiliary ones.

NFEs. We define h_i as the linear projection of its node features $l(x_i)$ for any $x_i \in V$ with output dimension $d_h = 128$.

PFEs. By extending the absolute positional encoding in the vanilla Transformer [Vaswani *et al.*, 2017], the cyclic positional encoding (CPE) was proposed in [Ma *et al.*, 2021b], which enables Transformer to encode cyclic sequences (as our PDP solutions) more accurately. The PFE g_i with output dimension $d_g = 128$ are initialized by CPE as follows,

$$g_i^{(d)} = \begin{cases} sin(\omega_d \cdot \left|(z(i) \bmod \frac{4\pi}{\omega_d}) - \frac{2\pi}{\omega_d}\right|), & \text{if } d \text{ is even} \\ cos(\omega_d \cdot \left|(z(i) \bmod \frac{4\pi}{\omega_d}) - \frac{2\pi}{\omega_d}\right|), & \text{if } d \text{ is odd} \end{cases} \quad (2)$$

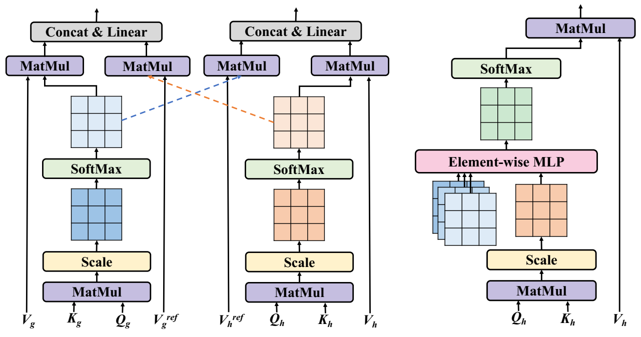
where superscript d of $g_i^{(d)}$ refers to the d-th dimension of g_i, and the scalar $z(i)$ as well as the angular frequency ω_d are defined in Eq. (3) and Eq. (4), respectively.

$$z(i) = \frac{i-1}{|V|} \frac{2\pi}{\omega_d} \left\lceil \frac{|V|+1}{2\pi/\omega_d} \right\rceil, \quad (3)$$

$$\omega_d = \begin{cases} \frac{3\lfloor d/3 \rfloor + 1}{d_g} \left(|V| - |V|^{\frac{1}{\lfloor d_g/2 \rfloor}}\right) + |V|^{\frac{1}{\lfloor d_g/2 \rfloor}}, & \text{if} d < \lfloor \frac{d_g}{2} \rfloor \\ |V|. & \text{otherwise} \end{cases} \quad (4)$$

According to [Ma *et al.*, 2021b], directly fusing the two sets of embeddings (i.e., $h_i + g_i$) may cause undesired noises to the vanilla self-attention. As shown in Figure 3(a), they thus proposed the DAC-Att in ways that each embedding set independently computes attention scores and shares the other *aspect* to learn dual-aspect representations. Different from it, we propose a simple and generic mechanism by incorporating a multilayer perception (MLP). As shown in Figure 3(b), besides the original self-attention scores (the orange squares), multiple auxiliary attention scores learned from other feature embeddings (the blue squares) are leveraged and fed into an

[4]$\mathcal{H}(t, K)$ is input to decoder and $f(\delta^*)$ is input to critic network (introduced later). Here we omit time step t for better readability.



(a) The **DAC-Att** [Ma *et al.*, 2021b] (b) Our **Synth-Att**

Figure 3: Comparison of attention modules for VRPs. The blue and orange squares are used to represent self-attention score matrices.

element-wise MLP, which allows it to synthesize heterogeneous attention relationships into comprehensive ones. We call it ***Synth**esis **Att**ention (**Synth-Att**)*. It is able to not only leverage more attention scores from various feature embeddings, but also achieve competitive performance to DAC-Att with less computation costs. Below, we present more details.

Auxiliary Attention Scores. In our N2S, PFEs are used to generate multi-head auxiliary attention scores as follows,

$$\alpha^{\text{aux}}_{i,j,m} = \frac{1}{\sqrt{d_k}} \left(g_i W_m^{Q_{\text{aux}}}\right)\left(g_j W_m^{K_{\text{aux}}}\right)^T, \tag{5}$$

where $W_m^{Q_{\text{aux}}} \in \mathbb{R}^{d_g \times d_q}, W_m^{K_{\text{aux}}} \in \mathbb{R}^{d_g \times d_k}$ are trainable matrices for each head m. We set $m = 4$ and $d_q = d_k = d_g/m$.

Syn-Att. The Syn-Att is defined as follows,

$$\tilde{h}_i = \textbf{Syn-Att}(W^Q, W^K, W^V, W^O, \text{MLP}). \tag{6}$$

In specific, it first computes the multi-head self-attention scores $\alpha^{\text{self}}_{i,j,m}$ for NFEs based on the trainable matrices $W_m^Q \in \mathbb{R}^{d_h \times d_q}$ and $W_m^K \in \mathbb{R}^{d_h \times d_k}$ for head m using Eq. (7) as

$$\alpha^{\text{self}}_{i,j,m} = \frac{1}{\sqrt{d_k}} \left(h_i W_m^Q\right)\left(h_j W_m^K\right)^T. \tag{7}$$

Thereafter, the attention scores α^{aux} and α^{self} are fed into a three-layer MLP with structure $(2m \times 2m \times m)$ to compute the synthesized multi-head attention scores as follows,

$$\alpha^{\text{Synth}}_{i,j,1}, ..., \alpha^{\text{Synth}}_{i,j,m} = \text{MLP}\left(\alpha^{\text{self}}_{i,j,1}, ..., \alpha^{\text{self}}_{i,j,m}, \alpha^{\text{aux}}_{i,j,1}, ..., \alpha^{\text{aux}}_{i,j,m}\right). \tag{8}$$

The scores are then normalized to $\tilde{\alpha}_{i,j,m}$ through Softmax, which are further used to calculate the attention values for each head as Eq. (9). Finally, the outputs are given by Eq. (10) with trainable matrix $W^O \in \mathbb{R}^{md_v \times d_h}$ $(d_v = d_h/m)$.

$$\text{head}_{m,i} = \sum_{j=1}^{|V|} \tilde{\alpha}_{i,j,m} \left(h_j W_m^V\right), \tag{9}$$

$$\tilde{h}_i = \text{Concat}\left[\text{head}_{m,1}, ..., \text{head}_{m,i}\right] W^O. \tag{10}$$

N2S Encoder. We stack L (L=3) encoders, each of which is the same as the Transformer encoder, except that the vanilla multi-head self-attention is replaced with our multi-head Synth-Att and we use the same instance normalization layer as [Ma *et al.*, 2021b]. Note that the auxiliary attention scores $\alpha^{\text{aux}}_{i,j,m}$ are only computed once and shared among all stacked encoders to reduce the computation costs.

4.2 Decoder

The N2S decoder first adopts the max-pooling layer in [Wu *et al.*, 2021] to aggregate the global representation of all embeddings into each individual one as follows,

$$\hat{h}_i = \tilde{h}_i^{(L)} W_h^{Local} + \max\left[\{\tilde{h}_i^{(L)}\}_{i=1}^{|V|}\right] W_h^{Global}. \tag{11}$$

Node-Pair Removal Decoder. Given the enhanced embeddings $\{\hat{h}_i\}_{i=1}^{|V|}$ and the set $\mathcal{H}(t, K)$, the *removal* decoder outputs a categorical distribution over n requests for removal action. In specific, it first computes a score λ_i for each $x_i \in V$ indicating the closeness between node x_i and its neighbors as

$$\begin{aligned}\lambda_i = (\hat{h}_{\text{pred}(x_i)} W_\lambda^Q)(\hat{h}_i W_\lambda^K)^T + (\hat{h}_i W_\lambda^Q)(\hat{h}_{\text{succ}(x_i)} W_\lambda^K)^T \\ -(\hat{h}_{\text{pred}(x_i)} W_\lambda^Q)(\hat{h}_{\text{succ}(x_i)} W_\lambda^K)^T,\end{aligned} \tag{12}$$

where $\text{pred}(x_i)$ and $\text{succ}(x_i)$ refer to the predecessor and the successor nodes of x_i, respectively, and $W_\lambda^Q \in \mathbb{R}^{d_h \times d_q}$, $W_\lambda^K \in \mathbb{R}^{d_h \times d_k}$. We use multi-head technique to obtain $\lambda_{i,1}$ to $\lambda_{i,m}$. Then the decoder aggregates the scores for each pickup-delivery pair (i^+, i^-) based on a three-layer MLP$_\lambda$,

$$\begin{aligned}\tilde{\Lambda}_{(i^+,i^-)} = \text{MLP}_\lambda(\lambda_{i^+,1}, ..., \lambda_{i^+,m}, \lambda_{i^-,1}, ..., \lambda_{i^-,m}, \\ c(i), \mathbb{1}_{\text{last}(1)=i}, \mathbb{1}_{\text{last}(2)=i}, \mathbb{1}_{\text{last}(3)=i}),\end{aligned} \tag{13}$$

where the MLP structure is $(2m + 4,\ 32,\ 32,\ 1)$, scalar $c(i)$ counts the frequency of request (i^+, i^-) being selected for removal in the past K steps, and $\mathbb{1}_{\text{last}(t)=i'}$ is a binary variable indicating whether request i' was selected at the t-th last step. An activation layer $\hat{\Lambda} = C \cdot \text{Tanh}(\tilde{\Lambda})$ is then applied $(C = 6)$, followed by Softmax to normalize the distribution which is then used to sample a node pair (i^+, i^-) as *removal* action.

Node-Pair Reinsertion Decoder. Given a request (i^+, i^-) for removal, the *reinsertion* decoder outputs the joint distribution that reinserts the two nodes back to the solution. We first define two scores $\mu^{\text{p}}(x_\alpha, x_\beta)$ and $\mu^{\text{s}}(x_\alpha, x_\beta)$ for a node x_α indicating the degree of preference of accepting a node x_β as its new predecessor and successor nodes, respectively,

$$\begin{aligned}\mu^{\text{p}}\left[x_\alpha, x_\beta\right] = (\hat{h}_\alpha W_\mu^{Q_{\text{p}}})(\hat{h}_\beta W_\mu^{K_{\text{p}}})^T, \\ \mu^{\text{s}}\left[x_\alpha, x_\beta\right] = (\hat{h}_\alpha W_\mu^{Q_{\text{s}}})(\hat{h}_\beta W_\mu^{K_{\text{s}}})^T,\end{aligned} \tag{14}$$

where $W_\mu^{Q_{\text{p}}}, W_\mu^{Q_{\text{s}}} \in \mathbb{R}^{d_h \times d_q}$, and $W_\mu^{K_{\text{p}}}, W_\mu^{K_{\text{s}}} \in \mathbb{R}^{d_h \times d_k}$. Again, we use multiple heads. Based on the scores, the decoder predicts the distribution of reinserting node i^+ after node j, and reinserting node i^- after node k using MLP$_\mu$,

$$\begin{aligned}\tilde{\mu}[j, k] = \text{MLP}_\mu(\mu_1^{\text{p}}[\text{succ}(j), i^+], ..., \mu_m^{\text{p}}[\text{succ}(j), i^+], \\ \mu_1^{\text{p}}[\text{succ}(k), i^-], ..., \mu_m^{\text{p}}[\text{succ}(k), i^-], \\ \mu_1^{\text{s}}[j, i^+], ..., \mu_m^{\text{s}}[j, i^+], \mu_1^{\text{s}}[k, i^-], ..., \mu_m^{\text{s}}[k, i^-]),\end{aligned} \tag{15}$$

where the MLP structure is $(4m,\ 32,\ 32,\ 1)$. Note that here $\text{pred}(\cdot)$ and $\text{succ}(\cdot)$ should be considered in the new solution where nodes i^+, i^- have already been removed. Afterwards, $\hat{\mu} = C \cdot \text{Tanh}(\tilde{\mu})$ is applied and infeasible choices are masked as $-\infty$ before normalizing by Softmax. Finally, a node pair (j, k), as the *reinsertion* action, is sampled according to the resulting distribution to indicate the positions of reinserting the node-pair (i^-, i^+) back to the solution.

Algorithm 1 n-step PPO with CL strategy

Input: policy π_θ, critic v_ϕ, PPO clipping threshold ε, learning rate η_θ, η_ϕ, learning rate decay β, epochs E, batches B, mini-batch κ, training steps T_{train}, CL scalar ρ^{CL}

```
1: for $e = 1$ to $E$ do
2:   for $b = 1$ to $B$ do
3:     Generate training data $\mathcal{D}_b$ on the fly;
4:     Initialize random solutions $\{\delta_i\}$ to $\mathcal{D}_b$;
5:     Improve $\{\delta_i\}$ to $\{\delta_i'\}$ via $\pi_\theta$ for $T = e/\rho^{CL}$ steps;
6:     Set initial state $s_0$ based on $\{\delta_i'\}$ and Eq. (1); $t \leftarrow 0$;
7:     while $t < T_{\text{train}}$ do
8:       Get $\{(s_{t'}, a_{t'}, r_{t'})\}_{t'=t}^{t+n}$ where $a_{t'} \sim \pi_\theta(a_{t'}|s_{t'})$;
9:       $t \leftarrow t + n$, $\pi_{old} \leftarrow \pi_\theta$, $v_{old} \leftarrow v_\phi$;
10:      for $k = 1$ to $\kappa$ do
11:        $\hat{R}_{t+1} = v_\phi(s_{t+1})$;
12:        for $t' \in \{t, t-1, ..., t-n\}$ do
13:          $\hat{R}_{t'} \leftarrow r_{t'} + \gamma \hat{R}_{t'+1}$;
14:          $\hat{A}_{t'} \leftarrow \hat{R}_{t'} - v_\phi(s_{t'})$;
15:        end for
16:        Compute RL loss $J_{RL}(\theta)$ using Eq. (18) and
           clipped critic loss $L_{BL}(\phi)$ using Eq. (19);
17:        $\theta \leftarrow \theta + \eta_\theta \nabla J_{RL}(\theta)$;
18:        $\phi \leftarrow \phi - \eta_\phi \nabla L_{BL}(\phi)$;
19:      end for
20:    end while
21:  end for
22:  $\eta_\theta \leftarrow \beta\eta_\theta$, $\eta_\phi \leftarrow \beta\eta_\phi$;
23: end for
```

4.3 Training Algorithm

As presented in Algorithm 1, the training algorithm for N2S is the same as [Ma *et al.*, 2021b], namely the proximal policy optimization with a curriculum learning strategy. It learns a policy π_θ (our N2S) with the help of a critic v_ϕ as follows.

Critic Network. Given the embeddings $\{\tilde{h}_i^{(L)}\}_{i=0}^{|V|}$ from policy network π_θ, the critic network first enhances them by a vanilla multi-head attention layer (with $m = 4$ heads) to get $\{y_i\}_{i=0}^{|V|}$. The enhanced embeddings are then fed into a mean-pooling layer in [Wu *et al.*, 2021] to aggregate the global representation of all embeddings into each individual one as

$$\hat{y}_i = y_i W_v^{Local} + \text{mean}\left[\{y_i\}_{i=1}^{|V|}\right] W_v^{Global}, \tag{16}$$

where we use trainable matrices $W_v^{Local}, W_v^{Global} \in \mathbb{R}^{d_h \times \frac{d_h}{2}}$. Lastly, the state value $v(s_t)$ is output by a four-layer MLP in Eq. (17) with structure (129, 128, 64, 1). Here, $f(\delta^*)$ is added as an additional input to MLP$_v$.

$$v(s_t) = \text{MLP}_v\left(\max\left[\{\hat{y}_i\}_{i=1}^{|V|}\right], \text{mean}\left[\{\hat{y}_i\}_{i=1}^{|V|}\right], f(\delta^*)\right) \tag{17}$$

We train N2S for E epochs and B batches per epoch, where the training dataset $\mathcal{D}_b$ is randomly generated on the fly with the uniform distribution (line 3). Following [Ma *et al.*, 2021b], we introduce a similar CL strategy (in lines 4 to 6) to further ameliorate the training. In specific, it improves the randomly generated solutions $\{\delta_i\}$ to $\{\delta_i'\}$ by running the current policy π_θ for $T = e/\rho^{CL}$ steps. The improved solutions $\{\delta_i'\}$ with higher quality are then used to initialize the first state s_0. In such a way, the hardness level of neighborhood search is gradually increased per epoch following the idea of curriculum learning. The remaining algorithm follows the original design of the n-step PPO, where we present the reinforcement learning loss function in Eq. (18), and the baseline loss function of critic in Eq. (19), respectively. Here, we clip the estimated value $\hat{v}(s_t)$ around the previous value estimates using $v_\phi^{clip}(s_{t'}) = clip\left[v_\phi(s_{t'}), v_{old}(s_{t'}) - \varepsilon, v_{old}(s_{t'}) + \varepsilon\right]$.

Algorithm 2 N2S-A Inference

Input: Instance $\mathcal{I}$ with size $|V|$, policy π_θ, maximum step T

```
1: for $i = 1, ..., \lfloor \frac{1}{2}|V| \rfloor$ do
2:   $\mathcal{I}_i \leftarrow \mathcal{I}$;
3:   $\mathcal{A}_i \leftarrow$ RandomShuffle([flip-x-y, 1-x, 1-y, rotate]);
4:   for each augment method $j \in \mathcal{A}_i$ do
5:     $\varrho_j \leftarrow$ RandomConfig($j$);
6:     $\mathcal{I}_i \leftarrow$ perform augment $j$ on $\mathcal{I}_i$ with config $\varrho_j$;
7:   end for
8: end for
9: Solve all instances $\mathcal{I}_i$ in parallel with $\pi_\theta$ for $T$ steps;
10: return the best solution found among all $\mathcal{I}_i$
```

$$J_{RL}(\theta) = \frac{1}{n|\mathcal{D}_b|}\sum_{\mathcal{D}_b}\sum_{t'=t}^{t+n} min\left(\frac{\pi_\theta(a_{t'}|s_{t'})}{\pi_{old}(a_{t'}|s_{t'})}\hat{A}_{t'},\right.$$
$$\left. clip\left[\frac{\pi_\theta(a_{t'}|s_{t'})}{\pi_{old}(a_{t'}|s_{t'})}, 1-\varepsilon, 1+\varepsilon\right]\hat{A}_{t'}\right), \tag{18}$$

$$L_{BL}(\phi) = \frac{1}{n|\mathcal{D}_b|}\sum_{\mathcal{D}_b}\sum_{t'=t}^{t+n} max\left(\left|v_\phi(s_{t'}) - \hat{R}_{t'}\right|,\right.$$
$$\left.\left|v_\phi^{clip}(s_{t'}) - \hat{R}_{t'}\right|\right)^2. \tag{19}$$

4.4 Diversity Enhancement

To be more resistant to local minima, we further equip our N2S with an augmentation-based inference scheme, which leads to N2S-A in Algorithm 2. Such idea was originally explored in [Kwon *et al.*, 2020] for a neural *construction* method, and we extend it to an *improvement* one. The rationale is that an instance $\mathcal{I}$ can be transformed into different ones for searching while reserving the same optimal solution, e.g., rotating all locations of nodes by $\pi/2$ radian. For an instance of size $|V|$, our N2S-A performs $\lfloor \frac{1}{2}|V| \rfloor$ augments, each of which is generated by sequentially applying four preset invariant transformation operations with different orders and different configurations as listed in Table 1. Note that although the mentioned transformations are conducted on instances defined in the Euclidean space, we believe that such an idea has favourable potential to be also exploited in non-Euclidean space, as long as there are proper invariant transformations for coordinates in the target space. Meanwhile, we use $K = |V|$ for training and $K' = \lfloor \frac{1}{2}|V| \rfloor$ for inference. This is because we found that when a specific K in $\mathcal{H}(t, K)$ is used for training, a smaller K during inference can improve the diversity of the solutions, thus better performance.

Transformations	Formulations	Configurations
flip-x-y	$(x', y') = (y, x)$	perform or skip
1-x	$(x', y') = (1 - x, y)$	perform or skip
1-y	$(x', y') = (x, 1 - y)$	perform or skip
rotate	$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \cos\vartheta - y \sin\vartheta \\ x \sin\vartheta + y \cos\vartheta \end{pmatrix}$	$\vartheta \in \{0, \pi/2, \pi, 3\pi/2\}$

Table 1: Descriptions of the four invariant transformations.

5 Evaluation

We design experiments to answer the following questions:

1. How good is the proposed N2S approach against the baselines, including the state-of-the-art neural methods and the strong LKH3 solver? (see Table 3 and Table 4)
2. Can Synth-Att reduce computation costs while achieving competitive performance to DAC-Att? (see Table 5)
3. How crucial are the proposed learnable node-pair removal and node-pair reinsertion decoders for achieving an efficient search? (see Table 6)
4. Can our N2S generalize well to benchmark instances that are different from training ones? (see Table 7)

5.1 Setup

We evaluate N2S on PDTSP and PDTSP-LIFO with three sizes $|V| = 21, 51, 101$ following the conventions in [Kwon *et al.*, 2020; Ma *et al.*, 2021b], where the node coordinates of instances are randomly and uniformly generated in the unit square $[0, 1] \times [0, 1]$. For our N2S, the initial solution δ_0 is sequentially constructed in a random fashion. Our experiments were conducted on a server equipped with 8 RTX 2080 Ti GPU cards and Intel E5-2680 CPU @ 2.4GHz. The training time of our N2S varies with problem sizes, i.e., around 1 day for $|V| = 21$, 3 days for $|V| = 51$, and 7 days for $|V| = 101$, all of which are shorter than the baselines in Table 2.

Hyper-parameters. Our N2S is trained with $E = 200$ epochs and $B = 20$ batches per epoch using batch size 600. We set $n = 5$, $T_{\text{train}} = 250$ for the n-step PPO with $\kappa = 3$ mini-batch updates and a clip threshold $\epsilon = 0.1$. Adam optimizer is used with learning rate $\eta_\theta = 8 \times 10^{-5}$ for π_θ, and $\eta_\phi = 2 \times 10^{-5}$ for v_ϕ (decayed $\beta = 0.985$ per epoch). The discount factor γ is set to 0.999 for both PDPs. We clip the gradient norm to be within 0.05, 0.15, 0.35, and set the curriculum learning ρ^{CL} to 2, 1.5, 1 for the three problem sizes, respectively.

5.2 Comparison Evaluation

We compare our N2S with the state-of-the-art (SOTA) neural methods and the highly-optimized LKH3 solver.

Regarding the former, we consider the SOTA *improvement* method DACT [Ma *et al.*, 2021b][5] and the SOTA *construction* method Heter-AM [Li *et al.*, 2021b] (specially designed for PDPs). To make a fair comparison with our N2S-A (with the diversity enhancement), we upgrade Heter-AM to Heter-POMO, also given the known superiority of POMO to AM. In specific, we reserve the policy network in Heter-AM as the backbone while adopting the diverse rollouts and the data augmentation techniques in POMO [Kwon *et al.*, 2020] to leverage the advantages of them for the best performance. Each neural baseline is trained using the respective implementation code that is publicly available. For the upgraded Heter-POMO, we adapt and combine the model architecture from the original Heter-AM and the original POMO. The links to their original implementations, approximate training time for the size $|V| = 101$, and the used hyper-parameters are presented in Table 2. For other hyper-parameters, we follow the recommendation in their papers. Regarding the latter, LKH3 is a strong heuristic (as reviewed in Section 2) which is widely used as a baseline to benchmark neural methods in recent studies (e.g., [Ma *et al.*, 2021b; Hottung *et al.*, 2021; Li *et al.*, 2021c; Wu *et al.*, 2021]). We report its results with two settings of iterations, i.e., LKH (5k) and LKH (10k).

[5]We use the *insert* decoder which is the best for PDPs.

Method	Code	Training Time	Hyper-parameters
Heter-AM	online[6]	$\sim$ 20 days	train 800 epochs as per the original setting.
Heter-POMO	online[7]	$\sim$ 14 days	train 2,000 epochs as per the original setting.
DACT	online[8]	$\sim$ 10 days	$\xi^{CL} = 0.25, 1, 4$ for sizes $\lvert v\rvert = 21, 51, 101$, respectively; $n = 5$, $T_{\text{train}} = 250$ (same as ours); train 200 epochs (same as ours).

Table 2: Training details of the adopted neural baselines.

All baselines are evaluated on a test dataset with 2,000 instances, and we report the metrics of averaged objective values, averaged gaps to LKH3 (10K) and the total solving time. Note that it is hard to perform an absolutely fair time comparison between running Python codes on GPUs (neural methods) and running ANSI C codes on CPUs (LKH solver). Thus we follow the guidelines in [Accorsi *et al.*, 2022] to perform the *facilitate comparison* that lets each method make full use of the best settings on our machine. In particular, we report the time of LKH3 when running in parallel with 16 CPU cores and the time of each neural method when all 8 GPU cards are available (but do not need to be fully used).

Results on PDTSP. Table 3 shows the results on PDTSP. In the first group, we compare N2S with Heter-AM (*greedy* and *sampling*), and DACT. Compared to Heter-AM (5k), our N2S with only 1k steps attains lower gaps with less time for all sizes. Although DACT offers the best gap on PDTSP-21, its performance drops significantly as the problem size increases, partly because its decoder is less efficient than our node-pair *removal* and *reinsertion* ones when tackling larger-scale problems. Instead, our N2S achieves higher performance and consistently dominates DACT in terms of both the gaps and the time on PDTSP-51 and PDTSP-101. In the second group, our augmented N2S-A is compared to the upgraded Heter-POMO method with three variants[9]. It is shown

[6]https://github.com/Demon0312/Heterogeneous-Attentions-PDP-DRL

[7]https://github.com/yd-kwon/POMO

[8]https://github.com/yining043/VRP-DACT

[9]Heter-POMO (gr.), Heter-POMO-A (gr.), Heter-POMO-A (3k) refer to: greedily generate $|V|$ solution; greedily generate $|V|$ solutions with 8 augments; sample $|V| \times 3$k solutions with 8 augments.

Methods	PDTSP-21			PDTSP-51			PDTSP-101		
	Obj. Value	Gap to LKH	Total Time	Obj. Value	Gap to LKH	Total Time	Obj. Value	Gap to LKH	Total Time
LKH (5k)	4.563	0.00%	3m	6.866	0.06%	10m	9.443	0.16%	49m
LKH (10k)	4.563	0.00%	5m	6.862	0.00%	19m	9.428	0.00%	98m
Heter-AM (gr.)	4.655	2.02%	(0s)	7.333	6.86%	(1s)	10.348	9.76%	(2s)
Heter-AM (5k)	4.578	0.33%	(33s)	7.108	3.58%	(1.5m)	10.051	6.61%	(5m)
DACT (1k)	4.572	0.20%	(18s)	7.245	5.57%	(29s)	10.551	11.91%	(51s)
DACT (2k)	4.566	0.07%	(37s)	7.118	3.72%	(1m)	10.312	9.38%	(1.5m)
DACT (3k)	4.564	0.03%	(1m)	7.057	2.83%	(1.5m)	10.195	8.13%	(2.5m)
N2S (1k)	4.573	0.21%	(21s)	7.103	3.51%	(31s)	10.030	6.38%	(1m)
N2S (2k)	4.567	0.09%	(42s)	7.053	2.77%	(1m)	9.905	5.06%	(2m)
N2S (3k)	4.565	0.05%	(1m)	7.027	2.40%	(1.5m)	9.846	4.44%	(3m)
Heter-POMO (gr.)	4.634	1.56%	(0s)	7.168	4.45%	(1s)	10.060	6.70%	(2s)
Heter-POMO-A (gr.)	4.584	0.46%	(1s)	6.995	1.93%	(5s)	9.681	2.68%	(11s)
Heter-POMO-A (3k)	4.564	0.03%	(7m)	6.916	0.77%	(32m)	9.567	1.47%	(135m)
N2S-A (1k)	4.563	0.01%	(1m)	6.865	0.03%	(8m)	9.475	0.50%	(40m)
N2S-A (2k)	4.563	**0.00%**	(2m)	6.860	**-0.03%**	(16m)	9.427	**-0.01%**	(80m)
N2S-A (3k)	4.563	**0.00%**	(3m)	6.860	**-0.04%**	(24m)	9.409	**-0.20%**	(121m)

Table 3: Comparison results for solving PDTSP instances of sizes $|V| = 21$, 51, and 101.

Methods	PDTSP-LIFO-21			PDTSP-LIFO-51			PDTSP-LIFO-101		
	Obj. Value	Gap to LKH	Total Time	Obj. Value	Gap to LKH	Total Time	Obj. Value	Gap to LKH	Total Time
LKH (5k)	5.539	0.00%	1m	10.218	0.17%	8m	17.115	0.34%	33m
LKH (10k)	5.539	0.00%	3m	10.200	0.00%	16m	17.057	0.00%	67m
Heter-POMO (gr.)	5.636	1.75%	(0s)	10.540	3.33%	(1s)	17.583	3.08%	(2s)
Heter-POMO-A (gr.)	5.567	0.51%	(1s)	10.353	1.50%	(5s)	17.276	1.29%	(10s)
Heter-POMO-A (4k)	5.545	0.12%	(10m)	10.209	0.09%	(48m)	16.890	-0.98%	(180m)
N2S-A (1k)	5.539	**0.00%**	(3m)	10.144	**-0.55%**	(13m)	16.976	-0.47%	(54m)
N2S-A (2k)	5.539	**0.00%**	(6m)	10.137	**-0.62%**	(27m)	16.859	**-1.16%**	(107m)
N2S-A (3k)	5.539	**0.00%**	(9m)	10.135	**-0.64%**	(40m)	16.806	**-1.47%**	(161m)

Table 4: Comparison results for solving PDTSP-LIFO instances of sizes $|V| = 21$, 51, and 101.

that even with only 1k steps, our N2S-A attains a significantly smaller gap than all three Heter-POMO variants, by almost an order of magnitude. Although Heter-POMO-A (gr.) tends to be competitive with fast speed, the gap is hard to be further reduced by increasing inference time if we refer to Heter-POMO-A (3k). Moreover, our N2S-A (2k) keeps abreast of, or even slightly exceeds the strong LKH3 solver, achieving gaps of -0.03% and -0.01% with less time on PDTSP-51 and PDTSP-101, respectively. Those gaps are further reduced to -0.04% and -0.20% with more steps, i.e., 3k.

Results on PDTSP-LIFO. In Table 4, we report the results on PDTSP-LIFO. Due to the more constrained search space, DACT failed to work well (see Table 6). Therefore, we mainly compare our N2S-A with Heter-POMO (the best neural baseline in Table 3) and the LKH3 solver. As exhibited, the advantages of neural methods over LKH3 have been further enhanced on this harder problem, where our approach consistently outperforms Heter-POMO for all sizes. Compared to LKH3, our N2S-A with 3k steps presents superior performance again, and attains gaps of -0.64% and -1.47% on PDTSP-LIFO-51 and PDTSP-LIFO-101, respectively.

5.3 Ablation Evaluation

Effects of Different Encoding Methods. We replace the proposed Synth-Att in our N2S encoder with vanilla-Att and DAC-Att, respectively. Using only one GPU card, we report

Att. in Encoders	Dim.	# Param.(M)	Time(s)	Gap(%)
Vanilla-Att	64	0.18 (1.00×)	239 (1.00×)	4.66
DAC-Att		0.31 (1.72×)	285 (1.19×)	2.89
Synth-Att		**0.19 (1.06×)**	**255 (1.07×)**	**2.88**
Vanilla-Att	128	0.72 (1.00×)	322 (1.00×)	3.92
DAC-Att		1.25 (1.73×)	400 (1.24×)	2.43
Synth-Att		**0.76 (1.06×)**	**340 (1.06×)**	**2.40**

Table 5: Effects of different encoding methods.

the number of model parameters, time, and gaps for solving 2,000 PDTSP-51 instances with 3k steps in Table 5. As exhibited, Synth-Att attains slightly smaller gaps than DAC-Att with much fewer computation costs, and considerably lower gaps than vanilla-Att with slight extra computation costs.

Effects of Different Decoding Methods. To highlight the desirability of our two decoders, we replace the trainable decoders with hand-crafted ones (i.e., random and the ϵ-greedy with $\epsilon = 0.1$) while ensuring feasibility. We gather the results on PDTSP-51 and PDTSP-LIFO-51 with 3k steps in Table 6 where mark '✓' means our proposed decoder is used (the one without augments) and mark '✗' means the decoder in the parentheses is used instead. As revealed, the methods of retaining at least one trainable decoder always attain much lower gaps than the ones equipped with only hand-crafted decoders. The method with both trainable decoders (i.e., N2S)

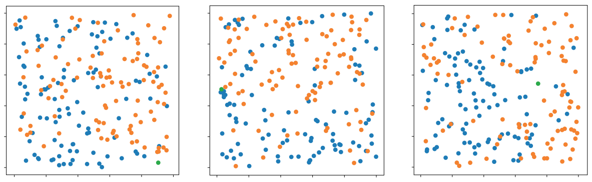

(a) N201P3 (b) N201P6 (c) N201P7

Figure 4: Example instances from [Renaud *et al.*, 2002].

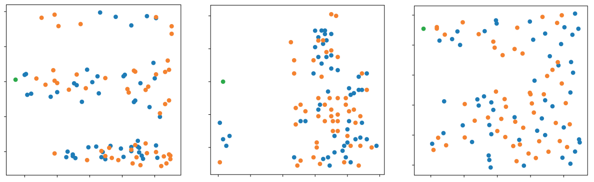

(a) brd14051_101 (b) pr1002_101 (c) fnl4461_101

Figure 5: Example instances from [Carrabs *et al.*, 2007].

Removal Decoder	Reinsertion Decoder	Gap(%) on PDTSP	Gap(%) on PDTSP-LIFO
✗(random)	✗(random)	210.82	112.37
✗(random)	✗(ϵ-greedy)	18.03	17.87
✗(ϵ-greedy)	✗(random)	86.31	47.57
✗(ϵ-greedy)	✗(ϵ-greedy)	15.62	12.64
✓	✗(random)	43.12	17.75
✓	✗(ϵ-greedy)	3.54	3.88
✗(random)	✓	6.38	4.38
✗(ϵ-greedy)	✓	8.42	6.28
✗(DACT)	✗(DACT)	2.83	11.73
✓	✓	**2.40**	**0.64**

Table 6: Effects of different decoding methods.

Problem	$\|V\|$	Gaps to	Heter-POMO-A		N2S-A	
			Avg.	Best	Avg.	Best
PDTSP	101	Opt.	1.64%	1.46%	**0.08%**	**0.00%**
	201		9.71%	8.66%	**2.82%**	**2.19%**
PDTSP-LIFO	$\leq$101	B1	10.06%	9.19%	**0.85%**	**-0.18%**
		B2	11.46%	10.58%	**2.11%**	**1.06%**

Table 7: Generalization performance on benchmark instances.

achieves the best performance. We also notice that DACT fail to solve PDTSP-LIFO well. This might be because its decoder considers removing and reinserting only one node instead of a pickup-delivery node pair in each action, which is a serious limitation for more constrained PDPs. For example, on PDTSP-LIFO, over 92% of the action space need to be masked for DACT, which leads to extremely low efficiency.

5.4 Generalization Evaluation

We further evaluate our N2S on benchmark instances, including all the ones from [Renaud *et al.*, 2002] for PDTSP and the ones with size $|V| \leq 201$ from [Carrabs *et al.*, 2007] for PDTSP-LIFO, which are largely different from our training ones, e.g., different sizes (i.e., 200 nodes) as shown in Figure 4 and different node distributions as shown in Figure 5. In Table 7, we report the best and the average gaps (with 10 independent runs) achieved by N2S-A and neural baseline Heter-POMO-A w.r.t. optimal solutions for PDTSP, or heuristic baseline B1 [Carrabs *et al.*, 2007] and B2 [Li *et al.*, 2011] for PDTSP-LIFO. It can be seen that our N2S significantly outstrips Heter-POMO in all cases. Without re-training or leveraging other techniques, this is fairly hard to achieve because machine learning often suffers from a mediocre out-of-distribution zero-shot generalization [Zhou *et al.*, 2021; Li *et al.*, 2021a]. The results also imply slight inferiority of our N2S to the LKH3 solver, given that LHK3 reports similar performance to the B2 baseline on those instances [Helsgaun, 2017]. Accordingly, we will focus on improving the out-of-distribution generalization for our N2S in future.

6 Conclusion

We present an efficient N2S approach for PDPs. It utilizes a novel Synth-Att to synthesize node relationships from various types of solution features and exploits the node-pair removal and reinsertion decoders to tackle the precedence constraint. Extensive experiments on PDTSP and PDTSP-LIFO verified our design, where N2S achieves state-of-the-art performance among existing neural methods. Further equipped with a diversity enhancement scheme, it even becomes the first neural method to surpass LKH3 on synthesized PDP instances. In future, we will 1) deploy N2S as a low-level agent in the hierarchical framework of [Ma *et al.*, 2021a] for dynamic PDPs, 2) combine N2S with a similar divide-and-conquer strategy in [Li *et al.*, 2021c] for much larger-scale instances, and 3) enhance the out-of-distribution generalization to exceed LKH3 on instances of arbitrary distributions.

Acknowledgments

This work was supported by the IEO Decentralised GAP project of Continuous Last-Mile Logistics (CLML) at SIMTech under Grant I22D1AG003. It was also supported in part by the National Natural Science Foundation of China under Grant 62102228, and in part by Guangdong Regional Joint Fund for Basic and Applied Research under Grant 2021B1515120078.

References

[Accorsi *et al.*, 2022] Luca Accorsi, Andrea Lodi, and Daniele Vigo. Guidelines for the computational testing of machine learning approaches to vehicle routing problems. *Operations Research Letters*, 50(2):229–234, 2022.

[Carrabs *et al.*, 2007] Francesco Carrabs, Jean-François Cordeau, and Gilbert Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading. *INFORMS Journal on Computing*, 19(4):618–632, 2007.

[Chen and Tian, 2019] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 32, pages 6281–6292, 2019.

[Fu *et al.*, 2021] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large TSP instances. In *AAAI Conference on Artificial Intelligence*, 2021.

[Helsgaun, 2017] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 2017.

[Hottung and Tierney, 2020] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *European Conference on Artificial Intelligence*, 2020.

[Hottung *et al.*, 2021] André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In *International Conference on Learning Representations*, 2021.

[Joshi *et al.*, 2019] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. arxiv preprint arxiv: 1906.01227, 2019.

[Kim *et al.*, 2021] Minsu Kim, Jinkyoo Park, and joungho kim. Learning collaborative policies to solve NP-hard routing problems. In *Advances in Neural Information Processing Systems*, volume 34, pages 10418–10430, 2021.

[Kool *et al.*, 2018] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[Kool *et al.*, 2021] Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. Deep policy dynamic programming for vehicle routing problems. arxiv preprint arxiv:2102.11756, ArXiv, 2021.

[Kwon *et al.*, 2020] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 21188–21198, 2020.

[Li *et al.*, 2011] Yongquan Li, Andrew Lim, Wee-Chong Oon, Hu Qin, and Dejian Tu. The tree representation for the pickup and delivery traveling salesman problem with LIFO loading. *European Journal of Operational Research*, 212(3):482–496, 2011.

[Li *et al.*, 2021a] Jingwen Li, Yining Ma, Ruize Gao, Zhiguang Cao, Andrew Lim, Wen Song, and Jie Zhang. Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*, 2021.

[Li *et al.*, 2021b] Jingwen Li, Liang Xin, Zhiguang Cao, Andrew Lim, Wen Song, and Jie Zhang. Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[Li *et al.*, 2021c] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *Advances in Neural Information Processing Systems*, volume 34, pages 26198–26211, 2021.

[Ma *et al.*, 2021a] Yi Ma, Xiaotian Hao, Jianye Hao, Jiawen Lu, Xing Liu, Tong Xialiang, Mingxuan Yuan, Zhigang Li, Jie Tang, and Zhaopeng Meng. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. In *Advances in Neural Information Processing Systems*, volume 34, pages 23609–23620, 2021.

[Ma *et al.*, 2021b] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative Transformer. In *Advances in Neural Information Processing Systems*, volume 34, pages 11096–11107, 2021.

[Nazari *et al.*, 2018] Mohammadreza Nazari, Afshin Oroojlooy, Martin Takáč, and Lawrence V Snyder. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9861–9871, 2018.

[Parragh *et al.*, 2008] Sophie N Parragh, Karl F Doerner, and Richard F Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

[Renaud *et al.*, 2000] Jacques Renaud, Fayez F Boctor, and Jamal Ouenniche. A heuristic for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 27(9):905–916, 2000.

[Renaud *et al.*, 2002] Jacques Renaud, Fayez F Boctor, and Gilbert Laporte. Perturbation heuristics for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 29(9):1129–1141, 2002.

[Savelsbergh, 1990] Martin WP Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47(1):75–85, 1990.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 6000–6010, 2017.

[Veenstra *et al.*, 2017] Marjolein Veenstra, Kees Jan Roodbergen, Iris FA Vis, and Leandro C Coelho. The pickup and delivery traveling salesman problem with handling costs. *European Journal of Operational Research*, 257(1):118–132, 2017.

[Wu *et al.*, 2021] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[Zhou *et al.*, 2021] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: A survey. arxiv preprint arxiv:2103.02503, ArXiv, 2021.